## 2.6 WEAPONS

Weapons are systems whose purpose is destroying or causing physical damage to a target. Weapons can consist of a delivery platform, a guidance method, and payload. Weapon deployment may include launching the weapon, guiding the delivery platform to the intercept point, determining the results of an intercept, and measuring the resultant damage, if any. Simulations of weapon deployment generally include explicit or implicit modeling of weapon delivery and a determination of resultant damage. Weapon delivery may include weapon-target intercept calculation.

Within SWEG, a weapons engagement is called a lethal engagement. It is a time-ordered sequence of mental events and possible physical events. A lethal engagement is an attack upon one player by another player using a weapon system. The various mental events determine whether a player will initiate a firing sequence during an engagement. The physical processes within the firing sequence may include a combination of the following: weapon launch, calculation of weapon-target intercept, recalculation of the intercept as necessary, determination of intercept results, abort determination, and abort results.

SWEG allows the user to create a weapon and assign it various capabilities including a probability of kill. All aspects of the weapon such as payload, weapon characteristics, delivery platform, onboard sensors, communications equipment, capabilities, susceptibilities and tactics are totally under user control. In reality, any weapon can be mounted onto any platform. Since the user has total control over player definition, weapons and platforms are not hard-wired together into “weapon systems”.

The flexibility within the user-created instruction files allows for the creation of weapon systems at varying levels of detail. At the lowest level of detail, a separate delivery platform is not created, the weapon is implicitly flown out to the target, and software table lookups and calculations compute the weapon-target intercept point, determine intercept status, and calculate the resultant damage. At a higher level of detail, weapon deployment is explicitly modeled; a separate platform which represents the delivery platform is created which disaggregates from its launcher at some point in the scenario.

### Weapon Delivery

Weapon delivery includes both propulsion and guidance. Some weapons, such as gravity bombs and bullets, are propelled from the launcher and the force of gravity determines their trajectory. Other weapons are self-propelled by rockets, jet engines or propellers. Weapons may also have different types of propulsion for different phases of their deployment. Ballistic missiles are launched into space by a rocket; however, they follow a ballistic or gravity trajectory following rocket-missile separation. Some weapons are implanted and do not have propulsion; land mines and tethered sea mines are examples.

Weapons can be propelled through the media of air, water and space; some weapons will transition between these various media. These various media impose speed constraints, and they may also attenuate or enhance weapon-related sensor performance. A submarine-launched ballistic missile (SLBM) will transition between water and air, air and space, and space and air. The three separate media through which it flies have different impacts on its speed.

SWEG does not have the capability of factoring media into speed calculations. The user must know the media that will be traversed and explicitly assign the speeds for each deployment phase. The user would assign the SLBM one speed while it traveled through the water, a second speed during rocket propulsion, and a third speed during the ballistic trajectory. The reentry vehicle nose cone would receive its own speed assignment for atmosphere reentry.

Weapons can have self-contained guidance control, they can receive guidance from an external source, or they can be unguided. Some missiles use self-contained controls that detect heat and guide the missile to the infrared radiation source. Some missiles are guided to their target via radio controls. Smart weapons, missiles and steerable bombs are equipped with laser or television guidance systems. Unguided weapons such as rockets and bullets fly ballistic trajectories once launched. SWEG can model all of these types of guidance.

SWEG implements weapon delivery with implicit and explicit flyouts. These two types of flyout permit the user to define a lethal engagement at varying levels of detail, and the type of flyout implemented determines how the weapon system's tactics, sensors, communications equipment, delivery platform and payload are modeled within the SWEG player structure.

Any weapon may be modeled with an implicit flyout in SWEG. If the user decides that it is not important for a weapon delivery platform to be detected or affected in any way during its transit to the target, it should be defined with an implicit flyout. For example, artillery bullets or other fire and forget weapons might be defined to have an implicit flyout. For implicit flyouts in SWEG, the weapon and its ordnance are defined within the player structure for the launcher; a separate player representing the delivery platform is not defined.

If the guidance details or weapon detectability are important to the user, a weapon may be given an explicit flyout in SWEG. Within SWEG, a player separate from the original launcher is created; this new or disaggregated player represents the delivery platform. In addition to defining a weapon and its associated payload within the player-structure for the delivery platform, the user must define any sensors, transmitters, communications equipment, and he deems necessary for the disaggregated player. The disaggregated player is given a perception of the launcher's target, but SWEG does not assume it must attack that target; the user may give it any tactics, independent of the point of view of its launcher.

If the user desires less detail, a weapon can be modeled in SWEG using an implicit flyout. SWEG provides a player definition framework which permits the user to define as much or as little detail as necessary to represent reality appropriately for the scenario. At a low level of detail, the user may define all weapons with implicit flyouts even though the actual weapons being modeled have onboard guidance systems and are detectable and attackable by other systems and weapons. This would be appropriate, for example, if the user knows there are no systems in the scenario capable of detecting the weapon during transit, or if he simply chooses not to allow that possibility. At a high level of detail, the user may choose an explicit flyout where the delivery platform is disaggregated from the shooter at some time during the scenario. This would be appropriate, for example, if the user wants the target of the weapon to be able to detect and evade it.

### Resultant Damage

It is the payload onboard the delivery platform that actually destroys the target or causes damage. This payload may be a conventional or nuclear explosive, a laser or a particle beam, or a chemical/biological agent. Non-explosive weapons cause destruction with focused energy while chemical/biological agents are dispersed through the air and cause physiological damage as they come into contact with human, animal or plant life. Resultant damage may include collateral damage; i.e., entities other than the original target may be damaged or destroyed. Antiaircraft artillery (AAA) may hit a friendly aircraft during an engagement, or the explosion of an ammunition depot may cause damage to a collocated railhead.

SWEG determines damage caused by a weapon via a user defined probability of kill table. This table depends on both the specific weapon and on the specific target. If is described in detail in the section on vulnerability. While the main purpose of the p(k) table is to estimate the damage against the primary target, the user can specify collateral elements belonging to the target within the p(k) table.

## 2.6.1 Functional Element Design Requirements

This section contains the design requirements necessary to implement the simulation of weapons in SWEG.

a. SWEG will provide the capability for a user to create a player with a weapon that can be either implicitly or explicitly fired at a target. For an implicit flyout, SWEG will compute the weapon-target intercept based on the weapon-target geometry and on a user-specified weapon speed profile. For an explicit flyout, SWEG will implement the creation of the disaggregated weapon at the appropriate time within the simulation and use reactive movement instructions to determine its flight path as it attempts to intercept the target. The disaggregated weapon will be modeled as a separate platform with user-specified attributes and characteristics.

b. SWEG will determine whether or not an implicit-flyout weapon can intercept its target based on relative target/weapon geometry and on user-specified weapon speed or movement path. If it cannot intercept the target, the weapon will abort and disappear from the scenario. If it can intercept the target, SWEG will calculate the resultant damage based on the condition of the shooter and the vulnerability of the target defined in terms of a probability of kill table for the weapon/target combination.

c. SWEG will provide the capability for a user to define the speed profile of ordnance as it implicitly moves out from the shooter to the target. The speed of the launcher at the time of launch will be added to these speeds, if the user so chooses. SWEG will use this data to calculate the approximate time and position of the intercept for implicit flyouts. SWEG will use this speed profile to calculate the maximum range and maximum flyout time for the weapon, and it will use these maximum values to determine if the intercept is possible or if an abort event should be scheduled.

d. SWEG will provide the capability for a user to specify characteristics for each implicit-flyout weapon type. The user options will include the following instructions:

   1. The dimension for the calculation of the flyout distance from the launch platform as either the two dimensional x-y plane or 3 dimensional. If a two-dimensional flyout is selected, SWEG will compute the time and position of the possible intercept by considering only the x-y plane differences between the weapon location and target. If a three-dimensional flyout is selected, SWEG will calculated the time and position of the possible intercept by considering the x-y and altitude differences between the weapon location and the target location.

   2. The time of the geometry calculations for the p(k) table lookup as either at launch or at intercept. If the p(k) table depends on relative weapon-target geometry, SWEG will use the positions at either the time of launch or the time of intercept as instructed.

   3. The weapon's mode of control after launch as either controlled or uncontrolled. In SWEG, controlled weapons are guided by the shooter after launch; the launcher has no control over uncontrolled weapons after launch and does not know when intercept occurs

   4. The option for self-destruction to be used when both the weapon and its platform are destroyed at intercept. If a weapon is defined as self-destruct, SWEG will cause both the weapon's platform and its parent platform to be removed from the exercise at either intercept or abort.

e. SWEG will provide the capability for a user to define two time delays for a weapon: the time delay between the decision to fire and the firing of the first round and the time delay between firing two rounds in a salvo. SWEG will use the time delay between the decision to fire and the start of movement to compute a total flyout time. If an abort event is scheduled, SWEG uses the total flyout time to compute the time of the abort event. SWEG will use the time delay between firing two rounds or a salvo to schedule subsequent weapon firing events for a weapon.

f. SWEG will provide the capability for a user to define the probabilities of kill for a given type of implicit-flyout weapon ordnance against a given type of target by creating a lookup table. This table is described in Section 2.6 Vulnerability.

g. SWEG will prevent a weapon from engaging more than the user-defined number of targets that can be engaged in parallel by its weapon type.

h. SWEG will allow ordnance to be added to a weapon system once a certain amount of ordnance has been consumed. The user can specify the number of extra rounds available, the amount of time needed to reload a given quantity of rounds, the threshold number of rounds that causes reloading to occur, and the quantity of rounds to be reloaded.

i. SWEG will prevent a weapon from exceeding any user-defined azimuth and elevation slew limits, both clockwise and counterclockwise, or limits on azimuth and elevation slew rates for its weapon type.

### 2.6.2 Functional Element Design Approach

A weapon type is declared under the element component of the player-structure in the TDB. Weapon is one of eight system categories within SWEG. The user has the option of defining resource categories for each system type. The appropriate resource categories for a weapon system are ordnance and future player. Ordnance is used for implicit flyouts, and future player is used for explicit flyouts. The user can specify the number of rounds for both of these resource categories. If the tactics within the player-structure select the weapon system, the user will define at least one capability for the weapon system.

Within SWEG, it is the element component of the player-structure that is attacked by the weapon. The player with the weapon has a perception of the target platform based upon the susceptibility of an element within that platform. The vulnerability of the element (and thus its parent components) to attack is discussed in Section 2.4.

#### Design Element 6-1: Implicit Flyout Weapons

Since a separate delivery platform is not created in an implicit flyout, the weapon and its ordnance are within the player-structure for its launcher; a separate player representing the individual rounds would not be created. SWEG uses the weapon speed profile table defined by the user and the future movement of the target to calculate the time and position of possible intercept as described in the following paragraphs.

In the TDB there is a table, called WPN-SPD-CAPABILITY, which is part of the weapons' type data base. This table defines speeds for the weapon for intervals of time. The speed is either a three- or two-dimensional speed, depending on the option selected in the WPN-CHARACTERISTICS table. If the 2D-FLYOUT option is selected, the altitude differences between the weapon location and target location are ignored. The integral of the time intervals and speeds in the WPN-SPD-CAPABILITY table determines the maximum range of the weapon. Consider this example of a speed capability table:

```
WPN-SPD-CAPABILITY
      DIMENSION 1  TIME (SEC)
                  0.      5.      20.     40.     60.
            AVG-SPD (M/SEC)
                  200.    1500.   2500.   1325.
END WPN-SPD-CAPABILITY
```

In this example, the maximum range of the weapon is:

(5 - 0) * 200 + (20 - 5) * 1500 + (40 - 20) *2500 + (60 - 40) * 1325 = 100,000 meters.

Note that this maximum range does not prevent the resource allocation tactics from firing at longer ranges. Also note that the target could be at a greater distance than this and if inbound, an intercept may still be possible. The WPN-TIME-DELAYS table has an entry in it that defines the initial time delay between the decision to shoot and the actual start of movement (implicit or explicit movement, depending on the detail). So, there is a constant

amount of time associated with the flyout that is dependent upon this number, which defaults to zero if not set by the user.

The best way to visualize the algorithm is to think about an expanding circle (two dimensions) or sphere (three dimensions) that expands at a rate determined by the WPN-SPD-CAPABILITY table. The expanding circle or sphere will either meet the future path of the target or the maximum range of the weapon will be reached before this intercept occurs. In the latter case there will be an abort scheduled at the game time equivalent to the maximum flyout time of the ordnance plus the shoot time delay, measured from the game time at which the decision to fire was made.

If the victim reactively maneuvers after the initial intercept determination, then a new event is scheduled to recalculate the intercept using the movement path. This new calculation can result in an earlier intercept, a later intercept, or an abort.

The expanding circle or sphere assumes best performance possible by the ordnance in determining the intercept. Since the flyout is implicit and there is no location associated with the ordnance during the flyout, the ordnance is able to perfectly look into the future to determine the minimum time to intercept. Note that this might not be the best intercept; depending on the kill probability table; i.e., the probability associated with this minimum time to intercept might be smaller than another probability associated with a longer intercept that could occur if the actual guidance techniques were used.

The intercept is iteratively approximated by checking for an intersection between the expanding circle or sphere and the straight lines or arcs of circles on the movement path.

### Design Element 6-2: Explicit Flyout Weapons

Within the TDB, an explicit flyout delivery platform is defined with a weapon system category and an ordnance resource category within its own player-structure. The delivery platform may have several system categories defined: thinker, sensor receiver, sensor transmitter, communications receiver, communications transmitter, mover, and/or disruptor. The weapon's delivery platform player will also have susceptibilities, capabilities, and tactics separate from the launcher player. The disaggregated player that is the weapon platform will also have its own weapon, which again may be implicit or explicit. As some point, however, one of the explicit flyout weapons must carry an implicit weapon; i.e., in the end, every weapon engagement in SWEG involves an implicit flyout weapon.

SWEG does not force the user to define a disaggregated player to be a weapon delivery platform and its payload. The user is free to create any player and define its tactics to do anything; e.g., for some unusual scenario, the user could disaggregate a submarine from a bomber or a bomber from a tank. SWEG only assumes two things about the newly created player. First, it is given a perception of the target platform's position. Second, the newly created player will be removed from the exercise if any of three events occur: the tracking sensor loses lock during a controlled weapon flyout, the target is removed from the exercise, or the element owning the weapon system is removed from the exercise. The newly created player will be removed from the exercise through an implicitly represented abort.

Within SWEG, the explicit flyout capability is initially defined by specifying a future player as a resource category. This future player represents the delivery platform for the weapon, and it allows for the creation of a new player at some time during the scenario. This player is created through disaggregation and is subordinate to its creating player. The delivery platform must then be completely defined within its own TDB player-structure.

SWEG cannot handle all situations when an explicit flyout might best represent the deployment of a weapon system. A beam-riding weapon uses its on-board sensor receiver to "ride" the signal created by the sensor transmitter on the launch platform. Typically, a new player would be created at some point during the exercise to represent the weapon delivery platform, and this new player would disaggregate from the launching player. In SWEG, a sensor receiver and transmitter cannot be paired across players. SWEG does not permit the user to define the sensor transmitter on one player (the launcher) and the sensor receiver on another player (the missile). A beam-riding weapon would have to be modeled within a single player; the launcher would be one platform and the delivery platform and payload would be the other platform. The sensor receiver and transmitter could then be paired since they are defined within the same player. By forcing both entities within a single player, an implicit flyout must be used.

## Design Element 6-3: Weapon Events

The physical aspect of modeling the movement of ordnance from the launching player to the target is controlled through weapon events. There are four weapon events: initial weapon launch, recalculation of the intercepts when the target has maneuvered, ordnance-target intercept calculation, and weapon abort. These four events share some command logic and operate together.

The initial launch event applies to both implicit and explicit flyouts. This event is triggered by a mental decision to launch ordnance or the continuation of ordnance launch in a salvo. Should the target change its path following launch, the previously calculated time-of-intercept is invalid and must be recalculated. When the target changes path, the recalculation event is triggered; and it then correctly accounts for the changes in the physical interactions between the two players and their locations. Several of these events occur during weapon flyout. The intercept event is the normal termination of a flyout either within the envelope of the weapon or at its fringe. The result can be a successful or unsuccessful intercept. An abort event is triggered when something goes wrong; i.e., the target is destroyed, the player controlling a flyout is destroyed, or the time that lock is lost for a controlled weapon is longer than the maximum coast time.

## Design Element 6-4: Initial Launch Event

The initial launch event consists of five steps: initialization, determination of arrival time, scheduling intercept determination, determination of intercept result, and event scheduling.

a. Initialization. If the target element selected by the mental event still exists, that target remains the object of the weapon. If it no longer exists, another target element on the same platform is chosen. Should no target elements exist, the ordnance will abort or be lost. Next, the ordnance is selected. It is possible that the mental decision event selected unavailable ordnance. It is also possible that

the ordnance may be depleted. If no ordnance remains, weapon event processing halts.

b. Determination of Arrival Time. The time that the ordnance-target intercept will occur is calculated. This step is also done during the recalculation of intercept event. At the initial calculation, the game time and the launch time are the same. During the recalculation event, the game time is later than the launch time. By using the weapon's speed profile provided by the user in the TDB and the predicted path of the target's platform, the SWEG software determines whether the intercept can occur. If specified by the user in the TDB, the SWEG software can also incorporate the velocity vector of the weapon platform into the ordnance's speed. If the ordnance's speed profile causes the ordnance to intersect the target's platform in time, an intercept is possible. If the intersection cannot occur, the intercept is not possible. In this case the time of flight is set to the maximum flyout time and a flag is set to signify the inability to intercept. If the intersection can occur, the intercept is computed in an iterative manner. The initial time step used in the iterative loop is 10-percent of the maximum flyout time of the ordnance. As the intercept point is determined, the time step is continually halved until it reaches a limit. The intercept point and time are finally determined. If the weapon has been defined as self-destruct in the TDB and the target is on or near the ground, it is possible that a crash may occur before the intercept event. If a crash occurs before intercept, the intercept will be considered unsuccessful.

c. Scheduling Intercept Determination. Based upon the results of the intercept time of arrival calculations, either an intercept or an abort event will be scheduled.

d. Determination of Intercept Result. If the p(k) calculations are specified to occur at launch, the determination of the kill probability is calculated at this point. Otherwise, the calculation is done at intercept. Should the previous step determine an abort, no kill probability will be calculated. Based upon the damage dimensions chosen by the user in creating the p(k) table in the TDB, a value is determined that will be used in a table lookup to determine the kill probability. This kill probability is saved for later use if the intercept actually occurs.

e. Schedule Events. If an intercept is possible and if the weapon's ordnance is flown out implicitly, an intercept event is scheduled. If an intercept is not possible and the flyout is implicit, an abort event is scheduled based on the maximum flyout time of the ordnance. These events are scheduled for the shooter in the case of a controlled flyout or for the environment in the case of an uncontrolled flyout.

When a future player is used in an explicit flyout, a player is created and given a perception of the targeted platform. Events are scheduled based on the created player's capabilities, initialization instructions in the SDB, and default system status from the TDB. These events are similar to those types of events that would normally be scheduled when the SDB is processed and original players are added to the exercise. The various events scheduled are based on system status for the various sensors, disruptors, and communications devices belonging to the player and on the tactics of the newly created player. For example, if the

player has lethal engagement tactics and if the type of the perceived platform is mentioned in the tactics implicitly with an ANYONE or explicitly through the platform's type, an entry will be made to the pending queue for a lethal engage queue add evaluation.

### Design Element 6-5: Recalculation of Intercept Event

This event consist of four steps: initialization, determination of arrival time, scheduling intercept determination, and determination or intercept result.

a. Initialization. This step is similar to that during the initial launch event described above except that the ordnance has already been selected. The target will be checked to ensure that it is still alive.

b. Determination of Arrival Time. This step is also the same as described for the initial launch event except that the game time and launch times are not the same. The possible results from this step are the same.

c. Scheduling Intercept Determination. Depending upon the intercept time of arrival calculated in the previous step, either an intercept event or an abort event is scheduled.

d. Determination of Intercept Result. This step is the same as it is for the initial launch event.

### Design Element 6-6: Weapon Intercept Event

This event consists of two steps: initialization and determination of intercept result. The initialization process is similar to initialization for the recalculation of intercept event. The following sub-steps occur in determining the result of the intercept.

a. If the user specified that the p(k) calculations would be done at intercept time, the kill probability is computed at this time. Otherwise, the p(k) was already computed at launch.

b. The kill probability is calculated the same way as described for a launch event. The geometry within the calculation uses the time of intercept instead of the time of launch.

c. If the target's element was defined as continuous, the kill probability used is an expected value and applied to the remaining amount of the element. Since the element cannot be killed, nothing more is done. Should the weapon be defined as self-destruct, the weapon's platform is destroyed and in turn the weapon's player will also leave the exercise.

d. If the target's element was defined as discrete a number of things can happen. A random draw is done which determines either a successful or an unsuccessful intercept. In the case of an unsuccessful intercept, nothing happens and the target continues in the exercise.

e. If the result of the random draw is a successful intercept, the integer counter for the platform that owns the target's element is decremented by one. If the counter becomes zero or less, the platform is destroyed. This destruction can in turn

cause aborts of missiles fired by the target's player as well as the abort of missiles that have already been fired at the now destroyed element.

f. If the weapon was defined by the user to be self-destruct, its platform and player will be destroyed and leave the exercise.

### Design Element 6-7: Weapon Abort Event

The weapon abort event's only step is to delete the data associated with the weapon firing. First it will delete the buffers that associate the weapon and the target. Next it will write out the history incidents to describe the abort as either a bad launch or as an in-flight abort. Third, the weapon will self-destruct in failure if it was defined as self-destruct. If the weapon was not self-destruct, the launcher will have an event scheduled to notice the results of the unsuccessful attack.

## 2.6.3 Functional Element Software Design

This section contains a table and six software code trees which describe the software design necessary to implement the requirements and design approach outlined above. Table 2.6-1 lists most of the functions found in the code trees, and a description of each function is provided. Figure 2.6-1 depicts the path from main to the top-level C++ function within the code for weapon events. Figures 2.6-2 through 2.6-6 contain the code trees for the individual weapons functions.

A function's subtree is provided within the figure only the first time that the function is called. Some functions are extensively called throughout SWEG, and the trees for these functions are in the appendix to this document rather than within each FE description. Within this FE, the functions in that category are MITRcontrol, lifbeg, and all member functions in the C++ class WhereIsIt.

Not all functions shown in the figures are included in the table. The omitted entries are trivial lookup functions (single assignment statements), list-processing or memory allocation functions, or C++ class functions for construction, etc.

TABLE 2.6-1. Weapons Functions Table.

| Function | Description |
|---|---|
| ailhit | schedules weapon hit on target event and updates lists |
| ailrip | schedules friendly players to notice death from attack |
| badpro | processes a randomly failed thinker/processor system |
| BaseHost::Run | runs all steps |
| begone | drops a perception block structures for outdated target |
| damage | assesses weapon attack results against target |
| delswt | deletes engaged weapon buffer block from various lists |
| delweb | drops the weapon event buffer from the snr-wpn-tgt buffer list and the weapon status list as well as recycle dynamic memory |
| diefri | deletes friendly perception structures |
| diesub | informs commander on a command chain when no operational subordinates are available |

TABLE 2.6-1. Weapons Functions Table. (Contd.)

| Function | Description |
|---|---|
| haltit | deletes movement data structures associated with platform |
| hitbeg | begins the weapon intercept process |
| hitend | ends the weapon intercept process |
| immol8 | performs self-destruction of platform and possibly parent player |
| injcom | deletes destroyed comm system |
| injsnr | deletes destroyed sensor system |
| injure | deletes systems belonging to an element |
| jetind | recycles indirect list entries and possibly top list |
| KILLperception | deletes a sensor-derived perception |
| lifbeg | begins a new player's life |
| lifcdr | processes weapon firing effects for newly created players |
| limevent | eliminates unneeded event nodes data structure |
| limpendq | eliminates unneeded pending queue data structures |
| main | controls overall execution |
| MainInit | initiates processing |
| MainParse | controls parsing of user instructions |
| MITRcontrol | controls emitter system status changes |
| mutil8 | deletes data structures associated with a platform |
| program | controls execution of all steps except bootstrap |
| redwood | adds new entry to or removes top entry from leftist tree |
| RNDMurn | determines a uniformly distributed random number |
| sasin8 | performs bookkeeping involved with death of a player |
| semant | controls semantic processing of instructions |
| simnxt | controls runtime execution |
| simphy | controls processing of physical events |
| simul8 | controls semantic processing of runtime instructions |
| srhpro | searches table for interval containing a specific value |
| TAccDirect::AddItem | adds an object pointer to the direct access structure |
| TAddrData::DelOccupant | deletes platform from the list of occupants at this address |
| TAddrData::GetPotentialAcqs | returns the potential acquisition list |
| TAddrData::TryDeletion | tries to delete a node in the address tree |
| TExpendable::SearchForType | Returns a pointer to expendable matching selected type or a 0 if not found |
| TMaster::GetPlatform | retrieves a platform from the direct access list |
| TMaster::GetPlayer | retrieves a player from the direct access list |
| TMaster::GetRandomTable | retrieves a random number table |
| TMemory::Allocate | allocates permanent storage |
| TMemory::AllocTemp | allocates temporary storage |

TABLE 2.6-1. Weapons Functions Table. (Contd.)

| Function | Description |
|---|---|
| TMemory::Deallocate | deallocates a list of blocks by using the address within the provided pointer |
| TMemory::DoAllocate | allocates either permanent or temporary storage |
| TMemory::LLSTaddend2 | adds entry to end of a list |
| TMemory::LLSTlistofflist | searches one list using a second list |
| TMemory::LLSTmeld | adds an entry to list if not already there |
| TMemory::LLSTremove | returns a pointer to the block on the traversed list which matches the provided key |
| TMemory::LLSTsearch | searches a list |
| TMemory::LLSTsearchhard | searches a list using extra parameters |
| TPathEntry::GetBeforePoint | looks up the path point prior to given time |
| trkfyr | schedules weapon firing after target is locked onto |
| TTable::SearchInt | searches a table for a specific integer |
| WhereIsIt::CalcHeading | determines heading for a platform given a time |
| WhereIsIt::CalcPosition | determines position for a platform given a time |
| WhereIsIt::CalcUnitVel | determines unit velocity for a platform given a time |
| WhereIsIt::CalcVelocity | determines local velocity vector for a platform given a time |
| wpnabo | controls weapon abort events |
| wpnarr | calculates the time and location of target intercept |
| wpnchz | chooses the target element for attack by a weapon |
| wpnfyr | processes weapon firing effects |
| wpngeo | performs geometry-related calculations for probability of kill determination |
| wpnhit | controls weapon intercept events |
| wpntgt | determines effectiveness of ordnance delivery on target and processes non-geometry related probability of kill data |
| yaeail | adds yet another entry to the scenario action item list |

```
main
    |-BaseHost::Run
        |-MainInit
            |-program
                |-MainParse
                    |-semant
                        |-simul8
                            |-simnxt
                                |-simphy
                                    |-wpnfyr
                                    |    |-wpnchz
                                    |    |-wpnarr
                                    |    |-wpntgt
                                    |    |    |-wpngeo
                                    |    |-lifbeg
                                    |-wpnchz
                                    |-wpnarr
                                    |-wpnhit
                                    |    |-wpnchz
                                    |    |-wpntgt
                                    |-wpnabo
```

FIGURE 2.6-1. Top-level Weapons Code Tree.

```
wpnfyr
    |-TMemory::Index2Ptr
    |-SysToTExpendable
    |    \-TMemory::Index2Ptr
    |-TExpendable::SearchForType
    |    |-TExpendable::GetNextPtr
    |    |    \-TMemory::Index2Ptr
    |    \-TExpendable::GetType
    |-TExpendable::GetNumber
    |-wpnchz
    |-TIncident::Capture
    |-TMemory::Ptr2Index
    |-TMemory::LLSTsearch
    |-TExpendable::GetType
    |-TMaster::GetPlatform
    |-DVector::DVector
    |-wpnarr
    |    |-TMemory::Index2Ptr
    |    |-WhereIsIt::CalcPosition
    |    |-WhereIsIt::CalcVelocity
    |    |-DVector::Putz
    |    |-isZeroEquiv
    |    |-DVector::operator-=
    |    |    |-DVector::Getx
```

FIGURE 2.6-2. wpnfyr Code Tree.

```
|    |    |-DVector::Gety
|    |    \-DVector::Getz
|    |-operator^
|    |-DVector::Norm
|    |-TMaster::GetResolution
|    |-operator-
|    \-DVector::GetLength
|-wpntgt
|    |-TMemory::Allocate
|    |    \-TMemory::DoAllocate
|    |         |-TMemory::GetBlockLength
|    |         |-CountMemOpns
|    |         |    \-TMaster::DebugOn
|    |         |-TMemory::Ptr2Index
|    |         |-ProgramStop::ProgramStop
|    |         \-TMemory::WriteSummary
|    |              |-TMemory::WordsUsed
|    |              |-TMemory::CalcWdsLeft
|    |              \-CountMemOpns
|    |-TMaster::GetPlatform
|    |-wpngeo
|    |    |-DVector::DVector
|    |    |-TMemory::Index2Ptr
|    |    |-WhereIsIt::CalcPosition
|    |    |-WhereIsIt::CalcVelocity
|    |    |-WhereIsIt::CalcUnitVel
|    |    |-DVector::Getx
|    |    |-DVector::Gety
|    |    |-operator-
|    |    |-DVector::GetHorizLength
|    |    |-operator^
|    |    |-operator/
|    |    |-DVector::Getz
|    |    |-dbg_sqrt
|    |    |-dbg_acos
|    |    |-WhereIsIt::CalcPitch
|    |    |-DVector::VecAng
|    |    |    |-operator^
|    |    |    |-operator*
|    |    |    |-DVector::DVector
|    |    |    |-DVector::GetLength
|    |    |    |-isZeroEquiv
|    |    |    \-dbg_atan2
|    |    |-operator-
|    |    |    |-DVector::DVector
|    |    |    |-DVector::Getx
|    |    |    |-DVector::Gety
|    |    |    \-DVector::Getz
```

FIGURE 2.6-2. wpnfyr Code Tree. (Contd.)

```
|   |   |-DVector::GetLength
|   |   \-dbg_asin
|   |       |-MathExcpt::MathExcpt
|   |       \-asin_c
|   |-TMemory::Index2Ptr
|   |-TTable::SearchInt
|   |   \-TMemory::Ptr2Index
|   |-TMemory::Ptr2Index
|   |-TMemory::LLSTsearchhard
|   |   |-TMemory::LLSTsearchhard
|   |   |   \-TMemory::Index2Ptr
|   |   \-TMemory::Index2Ptr
|   |-TMemory::LLSTsearch
|   |-TMemory::LLSTaddend2
|   |   |-TMemory::Index2Ptr
|   |   |-TMemory::Allocate
|   |   |-TMemory::AllocTemp
|   |   |   \-TMemory::DoAllocate
|   |   |-TMemory::Ptr2Index
|   |   \-TMemory::LLSTsearch
|   |-TMaster::DebugOn
|   \-srhpro
|-ailhit
|   |-TMaster::GetEnvironment
|   |   \-TMemory::Index2Ptr
|   |-TIncident::Capture
|   |-TMemory::LLSTmeld
|   |   |-TMemory::Index2Ptr
|   |   |-TMemory::LLSTsearchhard
|   |   |-TMemory::Allocate
|   |   |-TMemory::AllocTemp
|   |   |-TMemory::Ptr2Index
|   |   \-TMemory::LLSTaddend2
|   |-TMemory::LLSTsearch
|   |-TMemory::Ptr2Index
|   |-yaeail
|   |   |-TMemory::Index2Ptr
|   |   |-TMaster::GetGameTime
|   |   |-TMaster::DebugOn
|   |   |-TMessages::WriteMessage
|   |   |-TMemory::Allocate
|   |   |-TMemory::Ptr2Index
|   |   |-TMaster::PutScenrTree
|   |   |-redwood
|   |   |   |-TMemory::Index2Ptr
|   |   |   \-TMemory::Ptr2Index
|   |   |-TMaster::GetScenrTree
|   |   |   \-TMemory::Index2Ptr
```

FIGURE 2.6-2. wpnfyr Code Tree. (Contd.)

```
|    |    |-TMaster::GetPhase
|    |    \-TIncident::Capture
|    \-delweb
|         |-TMemory::LLSTremove
|         |    |-TMemory::Index2Ptr
|         |    |-TMemory::LLSTsearch
|         |    |    \-TMemory::Index2Ptr
|         |    \-TMemory::Deallocate
|         |         |-TMemory::Ptr2Index
|         |         |-TMemory::DeallocFront
|         |         |    \-TMemory::GetBlockLength
|         |         |-CountMemOpns
|         |         \-TMemory::RcylBlock
|         |              |-TMemory::Index2Ptr
|         |              \-TMemory::Ptr2Index
|         |-TMemory::Ptr2Index
|         \-TMemory::Index2Ptr
|-lifbeg
|-lifcdr
|    |-TMemory::Index2Ptr
|    |-TMemory::Ptr2Index
|    |-TMemory::LLSTsearch
|    |-TMemory::LLSTmeld
|    \-TMemory::LLSTremove
|-trkfyr
|    |-TMemory::Index2Ptr
|    |-TMemory::Allocate
|    |-TMemory::Ptr2Index
|    |-TExpendable::SearchForType
|    |-TExpendable::AddToNumber
|    |-TExpendable::GetKind
|    |-TExpendable::GetNumber
|    |-TExpendable::GetNextPtr
|    |-TExpendable::GetType
|    |-TIncident::Capture
|    |-yaeail
|    |-TMemory::LLSTmeld
|    \-diesub
|         |-TMemory::LLSTsearchhard
|         |-SysToTExpendable
|         |-TExpendable::GetNextPtr
|         |-TExpendable::GetType
|         |-TExpendable::GetNumber
|         \-TIncident::Capture
|-delweb
\-TMemory::Deallocate
```

FIGURE 2.6-2. wpnfyr Code Tree. (Contd.)

```
wpnchz
      |-TMemory::Index2Ptr
      |-TMemory::LLSTsearch
      |    |-TMemory::LLSTsearch
      |    |    \-TMemory::Index2Ptr
      |    \-TMemory::Index2Ptr
      |-TExpendable::GetNumber
      |-TMaster::GetPlayer
      |    \-TAccDirect::GetItem
      |         \-TAccDirect::GetItem
      |-TPlayer::IsAlive
      |-TIncident::Capture
      \-TMemory::Ptr2Index
```

FIGURE 2.6-3. wpnchz Code Tree.

```
wpnarr
      |-TMemory::Index2Ptr
      |-WhereIsIt::CalcPosition
      |-WhereIsIt::CalcVelocity
      |-DVector::Putz
      |-isZeroEquiv
      |-DVector::operator-=
      |    |-DVector::Getx
      |    |-DVector::Gety
      |    \-DVector::Getz
      |-operator^
      |-DVector::Norm
      |-TMaster::GetResolution
      |-operator-
      \-DVector::GetLength
```

FIGURE 2.6-4. wpnarr Code Tree.

```
wpnhit
    |-TMemory::Index2Ptr
    |-wpnchz
    |-TMemory::LLSTremove
    |    |-TMemory::Index2Ptr
    |    |-TMemory::LLSTsearch
    |    |    \-TMemory::Index2Ptr
    |    \-TMemory::Deallocate
    |-TMemory::Ptr2Index
    |-wpntgt
    |    |-TMemory::Allocate
    |    |    \-TMemory::DoAllocate
    |    |-TMaster::GetPlatform
    |    |-wpngeo
    |    |    |-DVector::DVector
    |    |    |-TMemory::Index2Ptr
    |    |    |-WhereIsIt::CalcPosition
    |    |    |-WhereIsIt::CalcVelocity
    |    |    |-WhereIsIt::CalcUnitVel
    |    |    |-DVector::Getx
    |    |    |-DVector::Gety
    |    |    |-operator-
    |    |    |-DVector::GetHorizLength
    |    |    |-operator^
    |    |    |-operator/
    |    |    |-DVector::Getz
    |    |    |-dbg_sqrt
    |    |    |-dbg_acos
    |    |    |-WhereIsIt::CalcPitch
    |    |    |-DVector::VecAng
    |    |    |    |-operator^
    |    |    |    |-operator*
    |    |    |    |-DVector::DVector
    |    |    |    |-DVector::GetLength
    |    |    |    |-isZeroEquiv
    |    |    |    \-dbg_atan2
    |    |    |-operator-
    |    |    |    |-DVector::DVector
    |    |    |    |-DVector::Getx
    |    |    |    |-DVector::Gety
    |    |    |    \-DVector::Getz
    |    |    |-DVector::GetLength
    |    |    \-dbg_asin
    |    |         |-MathExcpt::MathExcpt
    |    |         \-asin_c
    |    |-TMemory::Index2Ptr
    |    |-TTable::SearchInt
    |    |    \-TMemory::Ptr2Index
```

FIGURE 2.6-5. wpnhit Code Tree.

```
|    |-TMemory::Ptr2Index
|    |-TMemory::LLSTsearchhard
|    |    |-TMemory::LLSTsearchhard
|    |    |    \-TMemory::Index2Ptr
|    |    \-TMemory::Index2Ptr
|    |-TMemory::LLSTsearch
|    |-TMemory::LLSTaddend2
|    |    |-TMemory::Index2Ptr
|    |    |-TMemory::Allocate
|    |    |-TMemory::AllocTemp
|    |    |    \-TMemory::DoAllocate
|    |    |-TMemory::Ptr2Index
|    |    \-TMemory::LLSTsearch
|    |-TMaster::DebugOn
|    \-srhpro
|-hitbeg
|    |-prtenvintcp
|    |    |-TMemory::Index2Ptr
|    |    |-TIncident::Capture
|    |    |-TMemory::Ptr2Index
|    |    |-TMaster::GetPlatform
|    |    |-WhereIsIt::CalcVelocity
|    |    |-DVector::Getx
|    |    |-DVector::Gety
|    |    \-DVector::Getz
|    |-TMemory::Index2Ptr
|    |-TMemory::Ptr2Index
|    |-TMaster::GetPlatform
|    |-gnmatchast
|    |    |-TMemory::LLSTsearch
|    |    |-TMemory::Index2Ptr
|    |    |-to_ptr
|    |    \-ckgblcrit
|    |         |-ckgsycrit
|    |         |    |-TMemory::Index2Ptr
|    |         |    |-ck4cpcrit
|    |         |    |    |-TMemory::Index2Ptr
|    |         |    |    \-ckmodifier
|    |         |    \-ckmodifier
|    |         \-ckgwpcrit
|    |              |-TMemory::Index2Ptr
|    |              |-ck4cpcrit
|    |              \-ckmodifier
|    |-TMemory::LLSTsearchhard
|    |-damage
|    |    |-TMemory::Index2Ptr
|    |    |-dmgout
|    |    |    |-TIncident::Capture
```

FIGURE 2.6-5. wpnhit Code Tree. (Contd.)

```
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-TMemory::Ptr2Index
|   |   |   \-TMaster::GetPlatform
|   |   |-TMemory::Ptr2Index
|   |   |-RNDMurn
|   |   |   \-TMaster::GetRandomTable
|   |   |       \-TMemory::Index2Ptr
|   |   |-injure
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-injsnr
|   |   |   |   |-TMemory::Index2Ptr
|   |   |   |   |-TMemory::Ptr2Index
|   |   |   |   |-MITRcontrol
|   |   |   |   |-jetind
|   |   |   |   |   |-TMemory::LLSTremove
|   |   |   |   |   \-TMemory::Deallocate
|   |   |   |   |-TMemory::Deallocate
|   |   |   |   \-TMemory::LLSTremove
|   |   |   |-injcom
|   |   |   |   |-TMemory::Index2Ptr
|   |   |   |   |-MITRcontrol
|   |   |   |   |-TMemory::Deallocate
|   |   |   |   |-jetind
|   |   |   |   \-TMemory::LLSTremove
|   |   |   |-delswt
|   |   |   |-MITRcontrol
|   |   |   |-jetind
|   |   |   |-TMemory::Deallocate
|   |   |   |-TMemory::LLSTremove
|   |   |   \-badpro
|   |   |       |-TMemory::Index2Ptr
|   |   |       |-redwood
|   |   |       |-TMemory::LLSTremove
|   |   |       |-limpendq
|   |   |       |   \-TMemory::Deallocate
|   |   |       |-TMemory::Deallocate
|   |   |       |-TMemory::Ptr2Index
|   |   |       |-TIncident::Capture
|   |   |       \-limevent
|   |   |-mutil8
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-haltit
|   |   |   |   |-TMemory::Index2Ptr
|   |   |   |   |-TMemory::Ptr2Index
|   |   |   |   |-TMemory::Deallocate
|   |   |   |   |-TPathEntry::GetBeforePoint
|   |   |   |   \-TPathEntry::SetTDepart
|   |   |   |-TMemory::LLSTremove
```

FIGURE 2.6-5. wpnhit Code Tree. (Contd.)

```
|   |   |   |-TAddrData::DelOccupant
|   |   |   |   |-TMemory::LLSTremove
|   |   |   |   \-TAddrData::TryDeletion
|   |   |   |-TMemory::LLSTsearch
|   |   |   |-TMemory::Ptr2Index
|   |   |   |-limevent
|   |   |   |-yaeail
|   |   |   |-TMemory::Deallocate
|   |   |   |-TMemory::LLSTlistofflist
|   |   |   |-TIncident::Capture
|   |   |   \-TMaster::GetPlatform
|   |   |-ailrip
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-TMaster::GetPlayer
|   |   |   |-TPlayer::IsAlive
|   |   |   |-TMemory::Allocate
|   |   |   |-TMemory::Ptr2Index
|   |   |   \-yaeail
|   |   |-sasin8
|   |   |   |-TMemory::Index2Ptr
|   |   |   |-TMaster::GetEnvironment
|   |   |   |   \-TMemory::Index2Ptr
|   |   |   |-TMemory::Ptr2Index
|   |   |   |-diefri
|   |   |   |   |-TMemory::Ptr2Index
|   |   |   |   |-TMemory::LLSTsearch
|   |   |   |   |-TMemory::Deallocate
|   |   |   |   |-TMemory::LLSTremove
|   |   |   |   |-jetind
|   |   |   |   \-TMemory::Index2Ptr
|   |   |   |-KILLperception
|   |   |   |   |-TMemory::Ptr2Index
|   |   |   |   |-TMemory::Index2Ptr
|   |   |   |   \-begone
|   |   |   |       |-TIncident::Capture
|   |   |   |       |-TMemory::LLSTremove
|   |   |   |       |-TMemory::Deallocate
|   |   |   |       |-delswt
|   |   |   |       |-MITRcontrol
|   |   |   |       \-TMemory::Index2Ptr
|   |   |   |-TMemory::Deallocate
|   |   |   |-TMemory::Deallocate
|   |   |   |-limevent
|   |   |   |-TMemory::LLSTremove
|   |   |   |-redwood
|   |   |   \-limpendq
|   |   |-TExpendable::GetNextPtr
|   |   |   \-TMemory::Index2Ptr
```

FIGURE 2.6-5. wpnhit Code Tree. (Contd.)

```
|    |    |-TExpendable::GetNumber
|    |    \-TExpendable::SetNumber
|    |-TMemory::Deallocate
|    |-TPlayer::IsAlive
|    |-TMemory::Allocate
|    \-yaeail
|-delweb
|    |-TMemory::LLSTremove
|    |-TMemory::Ptr2Index
|    \-TMemory::Index2Ptr
|-hitend
|    |-TMemory::Index2Ptr
|    |-TPlayer::IsAlive
|    |-TMemory::Ptr2Index
|    |-TMemory::LLSTsearch
|    \-immol8
|         |-injure
|         |-mutil8
|         |-TMemory::Ptr2Index
|         |-ailrip
|         |-TMemory::Index2Ptr
|         \-sasin8
\-TMemory::Deallocate
```

FIGURE 2.6-5. wpnhit Code Tree. (Contd.)

```
wpnabo
|-TMaster::GetPlayer
|    \-TAccDirect::GetItem
|         \-TAccDirect::GetItem
|-TPlayer::IsAlive
|-TMemory::LLSTsearch
|    |-TMemory::LLSTsearch
|    |    \-TMemory::Index2Ptr
|    \-TMemory::Index2Ptr
|-TMemory::LLSTremove
|    |-TMemory::Index2Ptr
|    |-TMemory::LLSTsearch
|    |    \-TMemory::Index2Ptr
|    \-TMemory::Deallocate
|         |-TMemory::Ptr2Index
|         |-TMemory::DeallocFront
|         |    \-TMemory::GetBlockLength
|         |-CountMemOpns
|         |    \-TMaster::DebugOn
|         \-TMemory::RcylBlock
|              |-TMemory::Index2Ptr
|              \-TMemory::Ptr2Index
|-TMemory::Ptr2Index
```

FIGURE 2.6-6. wpnabo Code Tree.

```
|-TIncident::Capture
|-delweb
|    |-TMemory::LLSTremove
|    |-TMemory::Ptr2Index
|    \-TMemory::Index2Ptr
|-immol8
|    |-injure
|    |    |-TMemory::Index2Ptr
|    |    |-injsnr
|    |    |    |-TMemory::Index2Ptr
|    |    |    |-TMemory::Ptr2Index
|    |    |    |-MITRcontrol
|    |    |    |-jetind
|    |    |    |    |-TMemory::LLSTremove
|    |    |    |    \-TMemory::Deallocate
|    |    |    |-TMemory::Deallocate
|    |    |    \-TMemory::LLSTremove
|    |    |-injcom
|    |    |    |-TMemory::Index2Ptr
|    |    |    |-MITRcontrol
|    |    |    |-TMemory::Deallocate
|    |    |    |-jetind
|    |    |    \-TMemory::LLSTremove
|    |    |-delswt
|    |    |-MITRcontrol
|    |    |-jetind
|    |    |-TMemory::Deallocate
|    |    |-TMemory::LLSTremove
|    |    \-badpro
|    |         |-TMemory::Index2Ptr
|    |         |-redwood
|    |         |-TMemory::LLSTremove
|    |         |-limpendq
|    |         |    \-TMemory::Deallocate
|    |         |-TMemory::Deallocate
|    |         |-TMemory::Ptr2Index
|    |         |-TIncident::Capture
|    |         \-limevent
|    |-mutil8
|    |    |-TMemory::Index2Ptr
|    |    |-haltit
|    |    |    |-TMemory::Index2Ptr
|    |    |    |-TMemory::Ptr2Index
|    |    |    |-TMemory::Deallocate
|    |    |    |-TPathEntry::GetBeforePoint
|    |    |    \-TPathEntry::SetTDepart
|    |    |-TMemory::LLSTremove
|    |    |-TAddrData::DelOccupant
```

FIGURE 2.6-6. wpnabo Code Tree. (Contd.)

```
|   |   |   |-TMemory::LLSTremove
|   |   |   \-TAddrData::TryDeletion
|   |   |-TMemory::LLSTsearch
|   |   |-TMemory::Ptr2Index
|   |   |-limevent
|   |   |-yaeail
|   |   |-TMemory::Deallocate
|   |   |-TMemory::LLSTlistofflist
|   |   |-TIncident::Capture
|   |   \-TMaster::GetPlatform
|   |-TMemory::Ptr2Index
|   |-ailrip
|   |   |-TMemory::Index2Ptr
|   |   |-TMaster::GetPlayer
|   |   |-TPlayer::IsAlive
|   |   |-TMemory::Allocate
|   |   |-TMemory::Ptr2Index
|   |   \-yaeail
|   |-TMemory::Index2Ptr
|   \-sasin8
|       |-TMemory::Index2Ptr
|       |-TMaster::GetEnvironment
|       |   \-TMemory::Index2Ptr
|       |-TMemory::Ptr2Index
|       |-diefri
|       |   |-TMemory::Ptr2Index
|       |   |-TMemory::LLSTsearch
|       |   |-TMemory::Deallocate
|       |   |-TMemory::LLSTremove
|       |   |-jetind
|       |   \-TMemory::Index2Ptr
|       |-KILLperception
|       |   |-TMemory::Ptr2Index
|       |   |-TMemory::Index2Ptr
|       |   \-begone
|       |       |-TIncident::Capture
|       |       |-TMemory::LLSTremove
|       |       |-TMemory::Deallocate
|       |       |-delswt
|       |       |-MITRcontrol
|       |       \-TMemory::Index2Ptr
|       |-TMemory::Deallocate
|       |-TMemory::Deallocate
|       |-limevent
|       |-TMemory::LLSTremove
|       |-redwood
|       \-limpendq
\-TMemory::Deallocate
```

FIGURE 2.6-6. wpnabo Code Tree. (Contd.)

### 2.6.4 Assumptions and Limitations

- No reason for an aborted shot is provided.
- Weapons cannot fire at a location unless there is a perceived PLATFORM at that location.
- SWEG does not have the capability of factoring media into weapon speed calculations. The user must know the media that will be traversed and explicitly assign the speeds for each deployment phase/

### 2.6.5 Known Problems or Anomalies

None.